Assignment 3

CSE2215 Software Engineering Methods
2021/2022

Group 11b
Aleksandra Andrasz
Ciprian Stanciu
Matyáš Pokorný
Mike Raave
Milan de Koning

Tools

We have selected Pitest as a tool to compute the mutation score of our test suites.

Improved classes

We have identified these four classes with a mutation score less or equal to 70%:

- `UserModel`
- `Reservation`
- `Closure`
- `GroupServiceImpl`

Details

Mutation scores:	Mutation score before:	Mutation score after:
`UserModel`	20%	85%
`Reservation`	41%	66%
`Closure`	40%	92%
`GroupServiceImpl`	70%	95%

`UserModel`

We improved the mutation score on the `UserModel` class by adding more test cases for the `inRole` and `equals` methods. For the `inRole` method, we tested the method with an array without the required role and with an array that included both a required and a non-required role. This killed all mutants for this method. For the `equals` method, we added test cases to reach 100% branch coverage, which, since that method only deals with booleans also resulted in a perfect mutation score for that method. The only method left without a perfect mutation score in the class is the `hashCode` method. For this method it was quite hard to kill all mutants, since some were not quite equivalent, but produced a similar “semi-random” result (for example, replacing addition with subtraction still resulted in the method producing some seemingly random integer)

`Reservation`

We improved the mutation score on the `Reservation` class by adding test cases for the get methods and some other untested methods, such as the `createReservation`, `toModel`, `toString`, `setTimeFromRequest` and `fillOutTime` method. Each method needed just one test to cover all the mutants, only for the `fillOutTime` method we needed to make sure to cover the multiple paths to fully test the method. The only method left with unkilled mutants was the `hashCode` method. We tried to kill those mutants, but since it is difficult to get the result beforehand without using the `hashCode` method from the Objects class, since the method produces a semi-random result, we did not manage to kill many mutants there, just like with the `UserModel` method. This is why the mutation score is stuck at 66%.

`Closure`

The `Closure` object (part of the rooms microservice) is an embedded entity used to communicate room closure. Before we started mutation testing, it was at about 40% mutation coverage, mostly due to different tests that just so happened to use it. To improve the mutation score, a new test class was added which covered most of the common mutations. Biggest difference in mutation score made testing for stub implementations of properties. Tests for the `toModel` method, which converts the entity into its DTO, even though helped cover a whole lot of bugs, didn’t help the mutation score as much, as the only mutant that PIT produced for the method was `return null;`. The second biggest boost in mutation score was made by adding tests for the equality method, covering reflexivity, type equality and structural equality. The remaining 8% in mutation score can be largely attributed to the `hashCode` method. We did add tests to ensure that structural equality implied hash equality, but since we don’t really care (nor should we) about the precise implementation of this method, PIT mutants go undetected.

`GroupServiceImpl`

The `GroupServiceImpl` class is part of the Authentication microservice used to manage research groups. Before the changes, it had a mutation score of 70%. In order to improve it, we mostly improved tests, except for fixing the constructor of the InvalidData class as it did not set the reason. We added a test for verifying if the addGroup method correctly integrates the verifyUsers method that tests a group with users that are not in the database. Next, we improved tests for the getGroupInfo method, testing if a user who is only an admin can get information about a group, or if users that are only secretaries can get information about their groups. Finally, we improved the testing coverage for the addGroupMembers method, adding checks for groupIds that didn't exist in the database and making sure the verifyUsers method gets called.